ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

MASTER THESIS

Using Context Awareness to Improve Domain-Specific Named Entity Disambiguation

Author:
Matteo FILIPPONI

Supervisor:
Dr. Matteo ROMANELLO

Professor:
Prof. Frédéric KAPLAN

A thesis submitted in fulfillment of the requirements for the degree of Master in Computer Science

in the

Digital Humanities Laboratory
EPFL

June 19, 2017

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Abstract

In this project we designed and implemented a system based on the Learning To Rank framework to perform Named Entity Disambiguation (NED) of ancient author names and work titles being parts of canonical bibliographic citations. The data is made of abstracts extracted from modern publications in the context of Classical Studies.

We had to deal with domain specific challenges like the small set of available annotated data, the high level of ambiguity of the citations and a specific knowledge base which does not include the common properties of the knowledge bases usually used in state-of-the-art NED systems like Wikipedia.

Finally our system improved the already implemented baseline system and reached a F_1 score of 77.62% (+7.1%) and 71.88% accuracy (+10.2%). We also demonstrated how we can further improve the disambiguation by exploiting the co-occurrence probability of entities extracted from the corpus. With this method we improved our system by 6.8% in terms of accuracy on a sub-set of 59 documents.

Acknowledgements

I would like to thank my supervisor Matteo Romanello for his support and giving me very precious suggestions during the whole project. I also thank Maud Ehrmann for advising me on very specific subjects. A special thank to my parents for allowing me to continue my studies and to my girlfriend Martina for always supporting me.

Contents

Abstract **iii**

Acknowledgements **v**

1 Introduction **1**

2 State of the Art in Named Entity Disambiguation **3**
2.1 Task Definition . . . 3
2.2 Main techniques . . . 4
2.2.1 Candidate Entities Set Generation . . . 4
2.2.2 Candidate Entities Ranking . . . 5
2.2.3 NIL entities . . . 6
2.3 Previous Works . . . 6
2.3.1 Independent ranking methods . . . 6
2.3.2 Collective ranking methods . . . 7
2.3.3 Collaborative ranking methods . . . 8

3 Methodology **11**
3.1 Task Description . . . 11
3.1.1 Data & Annotation Schema . . . 11
3.1.2 Disambiguation . . . 12
Files format . . . 14
3.1.3 Pre-existing Software . . . 15
Knowledge Base . . . 15
Baseline . . . 15
3.1.4 Challenges . . . 16
3.2 Pre-processing . . . 17
3.3 Candidates Set Generation . . . 18
3.4 Candidates Ranking: Learning to Rank . . . 18
3.4.1 Ranking SVM . . . 19
Training . . . 20
Ranking function . . . 21
Interpretability . . . 21
3.5 Features . . . 23
3.5.1 String Similarity Features . . . 23
Exact Matching . . . 23
Fuzzy Matching . . . 23
Abbreviations and Acronyms . . . 24
3.5.2 Context Features . . . 24
Neighboring mentions similarities . . . 25
In-title mentions similarities . . . 25
TF-IDF cosine similarity . . . 25
3.5.3 Probability Features . . . 27

Entity prior probability 27
Mention ambiguity 27
3.5.4 NIL Features 27

4 Experiments and Evaluation **29**
4.1 Evaluation measures 29
4.2 Candidate Set Generation 30
4.3 Features Engineering 31
4.3.1 Per-type Ranking Function 31
4.3.2 Feature Combinations 32
4.3.3 NIL Features 33
4.4 Disambiguation Refinement through Co-occurrence Probability 33
4.5 Evaluation 35
4.5.1 Results 36
4.5.2 Error Analysis 37

5 Conclusion & Further Work **41**

Bibliography **43**

1 Introduction

Information Extraction (IE) is the task of extracting structured information from unstructured or semi-structured data. Initially developed for military purposes, this field is today present in many different domains. One reason of its popularity is the recent explosion of the information available under digitized form. As a consequence it is possible to automatically extract from text, usually written in the human language, the information by the application of Natural Language Processing (NLP) techniques.

One of the main applications of IE is the use of structured information to build Information Retrieval (IR) systems like for example internet search engines or library catalogs. Another application is to link the data structures between them, such created networks can then be analyzed, for example with the help of visualization tools, by domain experts but also by non-technical users.

One of the main sub-tasks of IE is the extraction of named entities, which can be itself splitted in two main tasks: Named Entity Recongnittion (NER) and Named Entity Disabniguation (NED). The goal of NER is recognizing if a word is a reference to a named entity while the goal of NED is to detect which entity it is referring to.

The goal of this project is to design and implement a NED system that can be able to disambiguate mentions of ancient authors and works. This module should improve an already existing NED system which is used in Romanello [24], which purpose is to automatically extract canonical references from Classical Studies documents.

2 State of the Art in Named Entity Disambiguation

2.1 Task Definition

In the field of Information Extraction, a named entity is any object, physical or abstract, that has a proper name. We can think of an entity as a category (e.g. Person, Country, City, Organization, etc.) and of a named entity as an instance of that category, for example *Pliny the Elder* is a named entity as it is an instance of Person. We need to distinguish between the text string "Pliny the Elder" and the named entity *Pliny the Elder*: the former is what is called a named entity mention and it is a reference, a pointer to the latter which is an abstract concept as it represents a unique identity, the referent of that mention.

The goal of the Named Entity Disambiguation (NED) task, sometimes called Named Entity Linking (NEL), is to determine the named entity which a particular mention is referring to and to assign to that mention the identifier of that entity. As a named entity is an abstract concept we need a way to uniquely identify the entities; to this purpose we use a Knowledge Base (KB) that contains the entities to which we need to link. As a result, the NED becomes the task of linking a mention with the corresponding entity identifier in a KB.

The main problem of such task is ambiguity. A named entity mention can refer to many possible entities and, conversely, an entity can be referenced by many different names, abbreviations or acronyms. A traditional approach is to extract semantic information from the context where the mention appears and from the document describing the entity (e.g. its Wikipedia article) to compare them. This highlights the importance of the KB, which is a key component of the disambiguation process. KBs like Wikipedia contain, in addition to the text describing the entry, in-going and out-going links to other articles, category tags, disambiguation pages and other meta-data that can be exploited to improve the efficiency of the task.

The NED task is closely related to the Word Sense Disambiguation (WSD) task. Instead of linking an ambiguous mention to a named entity, WSD deals with linking an ambiguous word, which has more than one meaning, to a sense or concept in a dictionary (e.g. WordNet). An important difference is that in WSD there is always an exact match between the word (its lemma) and the name of a possible meaning. For example the word *bass* can refer to either a fish, a musical instrument or a low frequency sound among others.

For the sake of clarity in the rest of this work we will refer to named entities as just "entities" and to named entity mentions as just "mentions".

2.2 Main techniques

Today's NED systems are typically divided into three modules. The first module creates a reasonably sized set of candidate entities for a given mention. The second module uses specific methods to identify the most probable entity from the set. The third module, which sometimes is included in the previous twos, handles the so-called NIL entities: when the correct entity that should be linked to a particular mention is not present in the KB.

2.2.1 Candidate Entities Set Generation

In general, when we want to disambiguate a mention, the first step is to generate a set of candidate entities. This is mainly due to the fact that knowledge bases usually contain several millions of entries making the job not acceptable in terms of computational complexity. Another reason is the reduction of the noise that is accomplished by excluding the entities that are likely to have any relation with the mention. In a few words, the goal of this first step is pruning the search space.

One important aspect to consider is the size of the candidates set. Depending on the method, a too large set can contain too much noise or it will be too slow to find the most likely correct entity. Another perhaps more important aspect is whether the candidates set contains or not the correct entity. This precision/recall trade-off is a delicate decision that has an important impact on the disambiguation performance.

The following techniques are used in the literature to generate the candidate entity set for a mention:

- **Name Dictionaries**

  The use of name dictionaries is the most used approach to generate candidate entities. In practice the dictionary is a key, value mapping where the key is a string representing a name and the value is a set of entities which can be referred by that key name. The dictionary is constructed off-line by iterating the entities of the KB, then, given an entity, all its name variants, abbreviations and other name forms are added as keys and the given entity is added to the set of entities of each key. When we want to generate the candidate entities for a mention, we just need to lookup in the dictionary for the surface form and get the related set of entities.

  A variant of this approach is to use approximate matching instead of exact matching when searching for the surface form in the dictionary. The advantage is an increase of recall but at the cost of introducing more noise in the candidates set.

- **Search Engines**

  Another method to generate the candidate entities is to exploit web search engines. Some NED systems used the Google API to submit the mention surface form as query, then they filtered out the Wikipedia links as candidates. Other NED systems directly used the search engine of Wikipedia.

- **Misspelling and Surface Form Expansion**

Depending on the origin of the data, names are likely to contain a varying degree of misspelling errors. Therefore in some cases it is necessary to cope with this problem before searching for candidate entities, especially when the text is the result of Optical Character Recognition (OCR).

A similar problem arises when the mention surface form is not present as a name variant of its entity in the KB, such reference is sometimes defined as an out-of-vocabulary (OOV) form. A typical case is when dealing with acronyms. In this case techniques like surface form expansion can be applied to try to reconstruct the full name by analyzing the text in which the mention appears. These methods are mainly grouped in two categories: rule-based and statistical.

2.2.2 Candidate Entities Ranking

Once the set of candidate entities is generated, we need a method to select the entity with the highest probability of being the correct match. Since there exist several approaches and techniques to deal with this ranking task, we give here an overview of these methods grouped into three categories.

- **Independent ranking methods**

  The characteristic of these methods is the assumption that the mention that need to be linked is independent from the other mentions that appear in the same document. The techniques that follow this approach are mainly based on the similarity between the context of the mention, extracted from the document in which it appears, and the document that represents the entity in the KB, typically its Wikipedia page.

- **Collective ranking methods**

  These methods assume that mentions appearing in the same document are semantically related as they are probably linked to the same topic. We call this notion the "topical coherence" between mentions of the same document. The consequence of this fact is that the disambiguation should be performed simultaneously for all the mentions of a single document. This technique allows to infer whether a candidate entity fits reasonably well with the candidate entities of the other mentions. Such joint inference is shown to be NP-hard [14], [12], [16] and therefore many systems use heuristic algorithms to compute approximate solutions.

- **Collaborative ranking methods**

  These methods are based on the assumption that similar mentions from different documents, and therefore having different contexts, could share their document information to augment their contexts so as to improve the detection of the similarity with a candidate entity context. A typical situation is when dealing with short texts like microblogs. Usually such methods are combined with collective ranking methods.

2.2.3 NIL entities

Not-In-List (NIL) entities are named entities that are not listed in the KB. If a mention refers to an entity that does not exist in the kb, then it should not be labeled, or, to be consistent, it should be labeled as a NIL. This problem is not always addressed in the literature as sometimes the data is assumed to not contain NIL entities. Systems tackling this problem often evaluate the system with and without NIL entities.

2.3 Previous Works

The NED problem has been well explored and there is a significantly amount of literature covering different approaches and techniques. In the following we will summarize the main methods that are used to deal with this problem. A more detailed survey of existing solutions is given in Shen et al. [25]. Since the rising of social networks, a vast amount of information has been available and exploitable. The main characteristic of this form of information (microblogs) is the shortness of the text and the high degree of ambiguity of mentions. The interest is such that it has become a field in its own right, in particular considerably efforts were made in addressing the NED task to tweets. A survey of the main techniques tackling this problem is given in Derczynski et al. [6].

2.3.1 Independent ranking methods

Bunescu et al. [2] proposed a method to extract and disambiguate named entities from Wikipedia. The extraction is based on heuristic rules applied to the text and leads to a dictionary mapping names to entities. The disambiguation problem is then casted as a ranking problem and two methods were tested. One ranking function they used is based on the cosine similarity between vector representations of the mention context and the entity document. The other ranking function was a kernelized version of Ranking SVM (Joachims [13]). Wikipedia categories and other semantic information (Wikipedia entity pages, redirection pages, categories, hyperlinks) like ancestor categories extracted from the entities Wikipedia taxonomy pages were included into the features to improve the similarity measure between mentions and entities. NIL entities are detected by fixing a threshold to the ranking function.

Dredze et al. [7] proposed a solution similar to [2] also based on the Ranking SVM algorithm to disambiguate named entities. They considered a KB independent approach but they showed that including optional Wikipedia features led to better results. They developed a rich feature set, which includes finite state machine to deal with name variations, estimation of entity popularity probability by the use of Google's PageRank algorithm [22] and several document similarity features. They also tried to combine features between them to try to capture eventual hidden correlations between them. Finally they included the detection of NIL entities in the learning framework so as to not handle them in a separate way as in [2].

2.3.2 Collective ranking methods

The work of Cucerzan [5] is one of the first that recognizes the semantic interdependence between entities in a document. Cucerzan proposed a system to perform both named entity extraction and named entity disambiguation. The disambiguation module is based on a vector space model (VSM). Each candidate of each mention of a single document is represented by a vector, which is constructed with the context and the category tags of the candidate's Wikipedia page. The context of an entity is constructed by extracting the references (hyperlinks) in the page and those appearing in the referenced pages. Then he aggregates all the in-document candidate's vectors of each mention to construct the document vector. The configuration of candidates that maximizes the similarity between their vectors and the document vector is chosen as the disambiguated set. The similarity is based on a local context compatibility score and the agreement between the categories of member candidates of that configuration. To avoid exponential complexity Cucerzan reduces the category agreement score to an inner-product between the candidate vector and the difference between the document vector and the candidate vector. This system obtained 88.3% accuracy against a baseline of 86.2% on a dataset of 350 Wikipedia articles (5'131 mentions). NIL entities were not included in the system.

Milne et al. [21] proposed a method to extract significant terms from text documents and link them to Wikipedia. The link detector is based on a machine learning approach, they used a classifier trained on linked or non-linked words from Wikipedia articles. The disambiguation module is based on two measures: commonness (prior probability of entity) and relatedness (the similarity between two entities). In contrast with [5] the relatedness measure is computed by exploiting the link structure of Wikipedia Milne et al. [20]. They define the coherence of a mention and a candidate as the relatedness of the latter with the entities of unambiguous mentions of the same document. They also introduced a measure of the quality of the unambiguous entities. A classifier is then trained on these features. NIL entities were not included in the system.

Kulkarni et al. [14] proposed a method to collectively disambiguate all mentions of a document by directly computing the joint optimization of all the combinations of mentions candidates. This approach does not depend on unambiguous mention like [21]. Given a mention and a candidate they compute the local similarity between them, which is the score given by the Ranking SVM algorithm trained with document text similarities between the mention context and the candidate Wikipedia page. Given a set of mentions and a set of candidates they compute the average of their similarity scores, called Node Potential, and the average of the relatedness between each pair of candidates, called Clique Potential. The goal is to infer the set of candidates that achieve the highest measure of agreement, which is the sum of Node Potential with Clique Potential, under the assumption that entities appearing in the same document are topically related. The relatedness between two candidate entities is computed by exploiting the categories and the hyperlink structure of their Wikipedia pages as in [21]. Since this method of collective disambiguation, as it is an instance of the Maximum Clique Problem, is NP-Hard, they tested a Linear Programming approach and an Hill-climbing technique to solve it. To deal with NIL entities they replaced the similarity score with a tunable parameter in the Node Potential, which represents the probability that the mention is actually a NIL entity. They reported 69.7% F_1 score on their own corpus populated with 107 documents collected from the web (17'200 mentions).

Hoffart et al. [12] modeled the collective disambiguation as an undirected weighted graph with mentions and candidates as nodes. Their framework is based on three main measures: the probability of an entity to be mentioned, the textual context similarity between mention and candidate, and the coherence between candidates for all the mentions in the document. Edges between mentions and candidates represent the context similarity while edges between candidates themselves represent the coherence. To disambiguate all mentions at once they devised a method called the coherence graph algorithm. To deal with exponential complexity the method is a greedy algorithm based on Sozio et al. [27] that compute a dense subgraph that should contain all mention nodes and exactly one mention-entity edge per mention. In addition to [14], to further reduce complexity and limit local optimal solutions they introduced a test based on prior probability and context similarity to fix a very probable candidate as true and to remove candidates that have very low scores. They obtained 81.8% accuracy on their own dataset based on CoNLL 2003 that consists of 1'393 Reuters newswire articles (34'956 mentions). In their experiments the system outperformed their re-implementations of [14] (72.8%) and [5] (51.0%). NIL entities were not included in the experiments. They used YAGO2 as knowledge base.

Han [9] proposed a method to model the global interdependence rather than the pairwise interdependence like in [14], [21], [5] between entities of a same document so as to capture indirect relations between entities. They model this global interdependence as a graph, called referent graph, similarly as done in [12]. The graph captures the similarity between a mention and an entity by using a BOW model and the similarity between entities by using the link structure of Wikipedia as proposed in [21]. In contrast with [12] they devised a purely collective algorithm to infer the mention-entity links by using evidence propagation on the referent graph. The initial evidence is represented by the mention-entity similarity. Then they use the referent graph structure to reinforce those probabilities by propagating them. Their propagation algorithm is similar to the Topic Sensitive PageRank algorithm [11]. They reported 73% F_1 score on the data set described in [14]. Experimental results show that the system outperformed [14] (69%), [21] (52%), [5] (45%) and [19] (37%). They did not addressed NIL entities.

More recently, Shen et al. [25] proposed a graph-based method, named KAURI, to disambiguate entities in tweets instead of addressing general text documents like in the previously described systems. Since tweets are very short texts, it's difficult to capture the context to help disambiguation as the information they contain could be insufficient. To mitigate this problem they used information about user interests, under the assumption that, given a user, it exists a topic interest distribution over the knowledge base entities. So their method still uses already seen features to rank candidate entities of a mention (entity popularity, textual context similarity, coherence between entities in a tweet) but it uses in addition the information about user interests to propagate the initial scores of the candidates. Their propagation algorithm is similar to the Topic Sensitive PageRank algorithm [11] and the algorithm used in [9]. They detected NIL entities by using a threshold like in [2]. They reported 85.8% accuracy on their own data-set composed by 1721 annotated tweets (2677 mentions).

2.3.3 Collaborative ranking methods

Chen et al. [3] elaborated three forms of collaborative ranking. The first one (micro collaborative ranking) is based on the assumption that by augmenting the query

with other similar queries (collaborators) the whole group could lead to a better disambiguation than by just using the query alone. The expanded query is then evaluated against each candidate by using a ranking function. The second method (macro collaborative ranking) is based on the assumption that there is not only one best ranking function for each query. Therefore, for each (query, candidate) pair, they used different ranking functions and then picked the best candidate by using a voting mechanism. The third method (micro-macro collaborative ranking) is a combination of the previous two. They used different supervised and unsupervised ranking functions for each (augmented-query, candidate) pair and showed that augmenting the queries led to improvements for each ranker. They included NIL entities in the rankers but they did not specify what technique they used to detect them. Experimental results showed that the third method performs better than the others by obtaining 83.7% (micro-averaged) accuracy on TAC-KBP2010 data set (2250 mentions).

Liu et al. [16] addressed the task of disambiguating mentions in tweets. The use of a collaborative framework is motivated by the lack of contextual information in tweets and the high level of ambiguity of the mentions [17]. To mitigate this problem they augmented the query by aggregating other similar mentions from other tweets, similarly to the micro collaborative ranking in [3]. The aggregation is also motivated by the necessity to deal with Out-of-vocabulary (OOV) mentions. Finally they used a collective ranking framework to disambiguate mentions of the augmented query against their candidates by integrating mention-entity, entity-entity and mention-mention similarities. They reported 71.1% F_1 score on a manually annotated data set [18] against two baselines: [19] (39.6%) and [18] (67.9%). NIL entities were not included in the system.

Similarly to [16], Guo et al. [8] addressed the NED problem to, more in general, microblogs (which also includes tweets). They used VSM to expand the post with the top-n most similar posts that contain the target mention. In contrast with [16], similar posts are not directly merged but they are used to construct a graph with candidates and posts as nodes. This choice is motivated by the fact that similar mentions of similar posts may not be co-referent. They modeled the problem as a graph by defining post-entity, entity-entity and post-post similarities. Then they used an iterative algorithm based on [31] and [28] to propagate entity labels and to jointly infer the mention-entity mappings, reminiscent of a collective ranking approach. Experimental results on their microblog data set show that the system outperforms the context-expansion based system. They also show that traditional NED systems performs less well on microblog data. NIL entities are not described in their experiments.

3 Methodology

3.1 Task Description

The goal of this project is to improve the Citation Disambiguation (CD) module of the automatic citation extractor pipeline described in Romanello [24] and shown in Figure 3.1. More specifically we address the Match Author/Work sub-module. The CD module represents the final step of the pipeline and it is placed after the Relation Detection and Citation Extractor modules. The input of the CD module is a named entity mention that could refer to either an ancient author or an ancient work. The goal of the module is then to identify to which author or work entity the mention is referring to and to link them together by attaching the entity identifier (represented in our KB) to the mention. Finally the output of the CD module is the annotated text, where each ancient and work mention is attached to an entity of the KB.

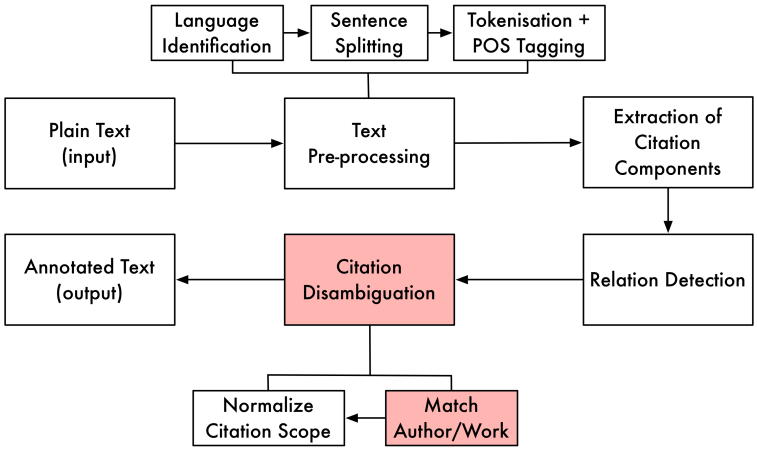


FIGURE 3.1: The pipeline used in Romanello [24]

In the rest of this section we describe the data and its format and also explain the schema used to annotate entity mentions. We also describe the pre-existing software, in particular about the KB and the baseline disambiguation module used in [24]. Finally we explain what are the main challenges that arise from this domain specific NED task.

3.1.1 Data & Annotation Schema

Our data set is composed by abstracts of scholarly works contained in *L'Année Philologique*[1] (APh), the standard bibliographical tool for research in Classical Studies published annually since 1924. As explained in Romanello [24] the APh contains an important number of canonical citations, which tend to follow a quite homogeneous style.

[1] http://www.annee-philologique.com

Furthermore the texts are written in different languages, namely English, French, German, Italian and Spanish.

To produce a gold data set, manually corrected, the data has been sub-sampled. Finally the data, which was also used for this project, consists of 366 documents extracted from the volume 75 (2004) of the APh. The language distribution is shown in 3.1.

TABLE 3.1: Language distribution over 366 documents

English	French	German	Italian	Spanish
65	123	39	107	32
17.8%	33.6%	10.6%	29.2%	8.8%

The annotation schema is composed by two elements: entities and relations. There are four types of entities that are extracted from the text as the output of the Citation Extractor module:

- *AAUTHOR*: A mention of this type refers to an ancient author, meaning that its surface form correponds to a name of an ancient author. An example of an *AAUTHOR* mention is "Ammianus".
- *AWORK*: In this case the mention refers to an ancient work. The surface form will correspond to a title of an ancient work. An example of an *AWORK* mention is "Aeneid".
- *REFAUWORK*: In this case the mention is a structured reference to an ancient work. It contains punctuation used to encode abbreviations of either an author, a work or a combination of the two. An example of a *REFAUWORK* mention is "Pliny, nat.". Note that "Pliny" refers to an ancient author and "nat." is an abbreviation of the title of one of his works.
- *REFSCOPE*: This entity captures the scope of a canonical citation and it is represented by numbers. For example in the citation "Lucr. 1, 62-79 e 3, 14-21" there are two *REFSCOPE* mentions, namely "1, 62-79" and "3, 14-21", which point to two sections (?) of a work of the author represented by "Lucr.".

The extracted entities are then linked between them by the Relation Detection module. Canonical citations are constructed by linking an entity referring to an ancient author or work (*AAUTHOR*, *AWORK*, *REFAUWORK*) to a *REFSCOPE*. Such relation is called *Scope*. However not all the mentions extracted from the documents have a scope meaning that there are not only canonical citations but also mentions referring to an author or a work without specifying a particular passage. As an example, in Figure 3.2 we can see an output of the extraction containing four Scope relations, two *AAUTHOR* mentions and two *AWORK* mention. This output corresponds to the input of the CD module.

3.1.2 Disambiguation

Once the mentions are extracted, the next step is to disambiguate them. As mentioned above the goal of the Citation Disambiguation module is to link canonical citations to their identifier. As explained in 2, traditional NED systems use Wikipedia

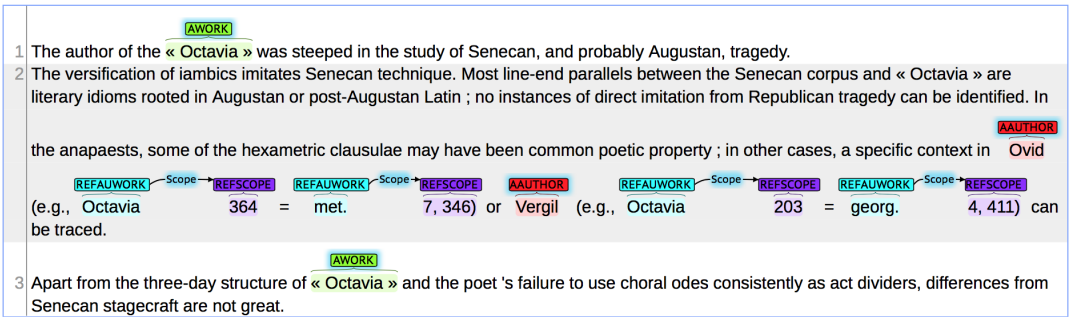


FIGURE 3.2: Extracted mentions of an APh document (75-03375) visualized in Brat

(or other derivatives like DBpedia or YAGO) as KB as it contains a lot of named entities and a lot of useful information about them. However, as explained in Romanello [24], the system has to automatically extract canonical citations and therefore it needs an identifier for each citable section of any ancient work but Wikipedia does not include all this information. Furthermore not all ancient authors and works are present in such KBs. As a consequence another solution had to be devised. The chosen one was to use the CTS (Canonical Text Services) protocol based on the URN (Uniform Resource Name) syntax to define the identifiers for the citations. As an example the mention "Vergil" of Figure 3.2 should be linked to `urn:cts:latinLit:phi0690` , while the mention "georg. 4, 411)" should be linked to `urn:cts:latinLit:phi0690.phi002:4.411`. The KB and its identifiers are further described in the next section.

As described in the previous chapter NIL entities are such named entities that are not contained in the KB. Since we decided to address such cases we needed a way to identify them. If a mention refers to a named entity which is not in our knowledge base, it should be linked to a special form of identifer representing a NIL entity. The NIL identifier is defined as `urn:cts:GreekLatinLit:NIL`. The motivations that led the inclusion of NIL entities in the system are discussed in the Challenges section.

The original Citation Disambiguation module described in [24] includes the normalization of the citation scope and the output labels of canonical citations (Scope relations) do contain the normalized passage (e.g. the `4.411` part of `urn:cts:latinLit:phi0690.phi002:4.411`). In this project we focused only on the pure disambiguation of ancient authors and works and as a consequence we did not include scopes in the identifiers. This is motivated by the fact that, in a *Scope* relation, when the name of the author or the work is disambiguated, appending the correct scope to the given CTS URN implies only the parsing and the normalization of the *REFSCOPE* mention and therefore this procedure could easily be performed in a separated module.

However we still need to consider *Scope* relations as they are an indication on the type of the entity that a particular mention is referring to. In the case that a mention has type *AAUTHOR*, if it is part of a Scope relation, meaning that it is related to a *REFSCOPE* mention, then this *AAUTHOR* mention is pointing to a work of the author represented in its surface form. This is the case of an *opus maximum*, when the title of the work does not need to be specified because it is implicitly known by the reader as it may be the only work written by that author or the more relevant.

For the reasons described above we have divided the mentions in four types:

- *AAUTHOR*
- *AAUTHOR_SCOPE*
- *AWORK*
- *AWORK_SCOPE*
- *REFAUWORK*

Files format

The annotated and labeled documents are serialized using the Brat[2] standoff markup format. For each text file containing the document, there is another file containing the extracted entities and relations as well as their labels. As an example, Figure 3.3 represents the annotation file for the document represented in Figure 3.2. At the top there the entities specified by id, type, offset and surface form. A the center there are the relations specified by id , type and the ids of the involved entities. At the bottom there are labels for each relation and for each standalone entity not involved in a relation.

```
T1  AAUTHOR 491 495 Ovid
T2  REFAUWORK 503 510   Octavia
T3  REFSCOPE 511 514    364
T4  REFAUWORK 517 521   met.
T5  REFSCOPE 522 529    7, 346)
T6  AAUTHOR 533 539 Vergil
T7  REFAUWORK 547 554   Octavia
T8  REFSCOPE 555 558    203
T9  REFAUWORK 561 567   georg.
T10 REFSCOPE 568 575    4, 411)
T11 AWORK 18 29 « Octavia »
T12 AWORK 629 640   « Octavia »

R1  Scope Arg1:T2 Arg2:T3
R2  Scope Arg1:T4 Arg2:T5
R3  Scope Arg1:T7 Arg2:T8
R4  Scope Arg1:T9 Arg2:T10

#0  AnnotatorNotes R4   urn:cts:latinLit:phi0690.phi002:4.411
#1  AnnotatorNotes R1   urn:cts:cwkb:1358.4386:364
#2  AnnotatorNotes R2   urn:cts:latinLit:phi0959.phi006:7.346
#3  AnnotatorNotes R3   urn:cts:cwkb:1358.4386:203
#4  AnnotatorNotes T6   urn:cts:latinLit:phi0690
#5  AnnotatorNotes T11  urn:cts:cwkb:1358.4386
#6  AnnotatorNotes T12  urn:cts:cwkb:1358.4386
#7  AnnotatorNotes T1   urn:cts:cwkb:568
```

FIGURE 3.3: Annotation file of an APh document (75-03375) in Brat standoff markup language

[2]http://brat.nlplab.org

3.1.3 Pre-existing Software

Knowledge Base

For the reasons mentioned before we need to use a specific KB. Our KB is made of authors and works. For each author the KB contains the following information: a unique CTS URN identifier, a list of names, a list of abbreviations and a list of work identifiers that point to the works of the given author. The list of names is actually a list of tuples, where in addition to the names it is also stored the language of the name. Similarly, for each work the knowledge base contains its identifier, a list of tuples (language, title), a list of abbreviations and the identifier of the author. The implementation and the population of the KB are discussed in detail in Chapter 3 of [24], in particular in section 3.6.

In Table 3.2 we can see an example of a KB entry for an ancient author, while in Table 3.3 the KB entry of its work. In Table 3.4 are shown some statistics about the KB. For this project We used the version ? of the KB[3].

TABLE 3.2: Author KB entry

id:	`urn:cts:cwkb:2907`
names:	[(la, Terpander)]
abbr:	[Terp.]
works:	[`urn:cts:cwkb:2907.9179`]

TABLE 3.3: Work KB entry

id:	`urn:cts:cwkb:2907.9179`
titles:	[(la, Fragmenta)]
abbr:	[Frg., Frag., Fr.]
author:	`urn:cts:cwkb:2907`

TABLE 3.4: KB statistics by authors and works

	Entries	Names/Titles	Abbreviations	Works avg.
Authors	1538	4831 (avg: 3.1)	773 (avg: 0.5)	3.4
Works	5199	10354 (avg: 2.0)	2376 (avg: 0.5)	-

Baseline

The current disambiguation module, which is also our baseline, is based on exact matching only [24]. Given a mention surface form, it searches through the KB for all the entities having a name or an abbreviation that matches the mention. The detection of an abbreviated mention is performed by analyzing the punctuation. In case of more than one matched entity, the system select the entity having the higher longest common string with the mention.

[3] https://github.com/mromanello/hucit_kb

3.1.4 Challenges

The first challenge we need to deal with is the high level of ambiguity of entity mentions in the Classics domain, and more specifically with regards to the task of disambiguating canonical references. In fact, titles of ancient works and names of authors are often abbreviated and may become very concise (i.e. be just a few characters long) (Example). In addition, there are several words that are commonly used in ancient work titles like "Epigrammata" or "Carmina" and also different authors share very similar names (Example with Aristophanes or Pliny). The table 3.5 taken from [24] illustrates the high level of ambiguity of title abbreviations of the ancient works in the KB.

TABLE 3.5: Top-10 ambiguous title abbreviations of ancient works in the knowledge base

Abbreviation	Meaning	Unique Works
epigr.	Epigrammata	96
carm.	Carmina	75
ep.	Epistulae	74
or.	Orationes	73
orat.	Orationes	71
epist.	Epistulae	66
hist.	Historiae	37
gramm.	Grammatica	36
ann.	Annales	16

Another challenge was to deal with the fact that the KB provides only a limited amount of information about the entities, especially it lacks of a textual description of the entity, an element widely used in state-of-the-art NED systems. As described in the previous chapter, such systems typically exploit the semantic information (article text, hyperlink structure, categories tag) contained in KBs like Wikipedia to help disambiguation by comparing candidate entities context to the textual context of the mentions [2, 7, 5, 21, 14, 12, 9, 25, 4, 16, 8] and/or to compare candidate entities against each other as done in collective and collaborative ranking based systems [5, 21, 14, 12, 9, 25, 4, 16, 8]. In our case the KB only contains the names/titles and abbreviations of ancient authors/works and pointers to their works/author and therefore we needed to devise another method to capture context information.

Furthermore the KB often contains incomplete information. Names and titles usually do not cover all the languages we need to deal with (en, es, de, fra, it) and sometimes the keyword specifying the language is missing. Abbreviations are also incomplete and sometimes they are mixed with the names/titles. This noise increase the difficulty of the task.

Another problem we need to deal with is having a small amount of data. Because of the specificity of the task, there are no annotated dataset that could be used to train and test supervised techniques and this explains why in [24] an annotated dataset had to be produced. Since manual annotation is a very time-consuming task, it is very difficult to create large data-sets. The consequence of having a small data set is that it will be difficult for a supervised technique to learn how to generalize well. It is also more difficult to have a test set being enough representative of the train set.

Finally we want to include NIL entities in our system. This task is not often addressed in state-of-the-art NED systems. The goal of detecting NIL entities is to avoid labeling a mention to an entity in the KB while the correct entity does not exist in that KB. This is incorrect but more importantly, even if we do not care whether we label correctly or not a NIL entity, this introduce noise in the linked data which could mislead its interpretation or mistaken its usage. The collateral effect is that we increase the risk of a non-NIL mention of being labeled incorrectly as we introduce the possibility of being NIL and this could have an important impact on the precision of the system. Concisely the main difficulty when dealing with NIL entities is to properly tune the system to avoid labeling non-NIL mentions as NIL or labeling NIL mentions as KB entities.

3.2 Pre-processing

When dealing with text comparisons it is necessary to apply some pre-processing steps to normalize words, to remove unwanted words and punctuation etc. For example, the strings "Julius" and "julius" are not the same word for a computer. Pre-processing is a fundamental step in text analysis and it must be carefully evaluated.

Since we are dealing with highly ambiguous names and abbreviations that contain, in general, very few words, we had to carefully decide what kind of pre-processing steps were suitable and also in what order apply them.

For example, punctuation is in general stripped out from the text but in our case it often gives some strong hints about the nature of the word. In particular dots suggest that the string is probably an abbreviation and this is useful when matching *REFAUWORK* mentions as they can contain mixtures of abbreviated and non-abbreviated words. Commas are sometimes use to separate the author name from the work title. However we noticed a significant amount of inconsistent punctuation notations in the data and therefore we decided to only use dots as helpers for the string matching.

In Table 3.6 we listed some real examples of mentions and the effect of each single pre-processing step on them. The pre-processing has been applied to the mention surface forms and to the names/titles and abbreviations of the KB entities.

TABLE 3.6: The effect of the pre-processing steps

Pre-processing step	Input	Output
Remove numbers	1 Kor	Kor
Remove roman numbers	Innocenzo I	Innocenzo
Strip accents	Parménide,	Parmenide,
Remove apostrophe words	l' « Isagoge »,	« Isagoge »,
Remove single characters	Teofilatto e	Teofilatto
Remove punctuation (except dots)	(Apol.	Apol.
Lowercase	Cicero	cicero
Remove stop-words	Avitus of Vienne,	Avitus Vienne,

3.3 Candidates Set Generation

According to state of the art NED systems best practices described in the previous chapter, we addressed first the generation of a set of candidate entities for each mention. The goal of this step is to produce, for each mention, a reasonably size-limited set of candidates that does contain the true entities.

As described in section 2.2.1 there are two techniques that are mainly used to generate the candidates set: search engines based and name dictionary based. Since we are dealing with names and titles that come from a specific domain and our KB contains, in general, more than one name variation for each entity, we decided to use the name dictionary based method.

Since a mention refers to either an author or a work we had to build two separate dictionaries for each type. Furthermore, when we match a mention surface form against a name or an abbreviation we use different similarity functions and therefore we decided to use a different dictionary for the abbreviations. Finally we constructed 4 dictionaries mapping respectively author names, author abbreviations, work titles and work abbreviations to their identifiers. For example if there are two authors e_i and e_j that have one common abbreviation $abbr = a_{i,k} = a_{j,l}$, where $a_{i,k}$ is the k^{th} abbreviation of the entity e_i. Then the dictionary D_{author_abbr} will contain the mapping $D_{author\_abbr}[name] \rightarrow \{e_i, e_j\}$.

Once we have built the dictionaries, the candidates set is generated differently depending on the mention type as described in the following:

- *AAUTHOR* not linked to *REFSCOPE*: We search for a match in the D_{author_names} and D_{author_abbr} dictionaries. All the author entities mapped to the matched keys are added as candidates.
- *AAUTHOR* linked to *REFSCOPE*: As before, we search for a match in the D_{author_names} and D_{author_abbr} dictionaries. However the mention refers to a work, therefore we add as candidates all the works of the matched author entities.
- *AWORK*: We search for a match in the D_{work_titles} and D_{work_title} dictionaries. All the work entities mapped to the matched keys are added as candidates.
- *REFAUWORK*: Since the surface form of this type of mention could represent either an author name or abbreviation, a work title or abbreviation or a mix of them we need to search in all the dictionaries. If we match a key in the authors dictionaries, we add as candidates all the works of the mapped author entities.

The match between a mention surface form and a name/title or abbreviation of an entity depends on different factor. The functions we used to define such comparison are detailed in section 4.2.

3.4 Candidates Ranking: Learning to Rank

Once we generated a set of candidates for each mention we needed a method to rank them to select the most probable candidate entity. As mentioned in section 3.1.4, we cannot use the collective and collaborative techniques described in chapter 2 mainly because of the specificity of our KB. Therefore the chosen method belongs

to the independent ranking category, in particular we wanted to use a supervised technique so as to take advantage of the manually annotated data we dispose of.

The goal is to use the labeled data to train a model that could select the most probable entity to attach to a mention. More formally, the task can be formulated as follows: given a mention m and a list of candidate entities E_m, then we want to pick a candidate $e_i \in E_m | e_i = e^m$, where e^m is the true entity referenced by m. Note that we assumed that $e^m \in E_m$, which could be false. However the presence of the true entity in the candidate set depends on another module, having as one of its main goals to reach a sufficiently high recall.

Using a binary classifier seems the most natural way to solve this problem and some NED systems already used this approach [30, 3]. The model is trained as follows. For each $e_i \in E_m$ we generate the feature vector $\langle m, e_i \rangle$ and we label it as positive if $e_i = e^m$ and negative otherwise. However, as described in [25] such models suffer from two main drawbacks. The first is the unbalanced nature of the data caused by the fact there is only one positive sample and many negatives. The other is that they have to deal with the situation of having more than one positive sample when using the trained model.

To overcome these drawbacks, more recent NED systems used a learning to rank approach and in particular they used the Ranking SVM algorithm [7, 2]. The goal of a Learning to Rank framework is to learn a ranking function from lists of data samples where an order is specified. A typical application is to improve Information Retrieval systems. When applying this framework to the NED task the problem is reduced to ranking first the correct candidate entity, the order of the other candidates does not matter.

Based on the above considerations, we decided to employ the Ranking SVM algorithm to face our task. Since we did not find a Python library implementing such algorithm, we implemented it by using the SVM implementation of the sklearn Python library Pedregosa et al. [23]. In the following we explain how the algorithm works.

3.4.1 Ranking SVM

Ranking SVM is a supervised learning algorithm based on the well known SVM algorithm and it is used to solve ranking problems. It was originally published by Joachims [13] as a method to improve internet search engines. It is based on a pairwise ranking approach and its goal is to learn a ranking function from queries where results are given and following a certain ranking order. In this section we explain the adaption of such algorithm to the NED task.

The difference between the general use of the algorithm for IR systems and its use in NED systems is the labeling of the rankings. In IR systems the items that need to be ranked are, in general, distributed in different groups defining some partial order. For example 7 items could have ranks as $[3, 3, 2, 1, 1, 1, 0]$, where 3 is the highest rank. When using the algorithm in NED systems we only care about the correct entity and therefore we have only two groups of labels: the label for the correct entity and the label for all the incorrect candidates, e.g. $[1, 0, 0, 0, 0, 0, 0]$ in the case of 7 entities. Therefore only the candidate that is ranked first is considered.

The application of the Ranking SVM algorithm to the NED task is formulated as follows. For each mention m, its true mapping entity e_m and the set of candidates

E_m where $em \in Em$, we assume that exists a linear ranking function F_r such that the score given by $F_r(e_m)$ is greater than any $F_r(e_i)$ where $e_i \neq e^m$. More formally, the goal is to find a a ranking function $F_r(x) = xw$ that satisfies:

$$\forall m, \forall e_i \in E_m | e_i \neq e^m : F_r(e^m) > F_r(e_i) \tag{3.1}$$

However solving directly (3.3) is shown to be NP-hard [**nphard**]. The solution can be approximated with SVM techniques by introducing (non-negative) slack variables $\xi_{m,i}$ and minimizing the upper bound $\sum \xi_{m,i}$. The problem can then be formulated as:

$$\begin{aligned} &\text{minimize} \quad \|w\|^2 + C \sum \xi_{m,i} \\ &\text{subject to} \quad \forall m, \forall e_i \in E_m | e_i \neq e^m : F_r(e^m) \geq F_r(e_i) + 1 - \xi_{m,i} \end{aligned} \tag{3.2}$$

where C is a parameter to tune the trade-off between the margin size and the training error. If we re-arrange the constraints of the equation (3.2) as:

$$\forall m, \forall e_i \in E_m | e_i \neq e^m : F_r(e^m) - F_r(e_i) \geq 1 - \xi_{m,i} \tag{3.3}$$

we can see that finding an optimal ranking function is equivalent to a SVM classification of pairwise difference vectors. In the following we show how in practice the algorithm is used to perform the disambiguation of entities.

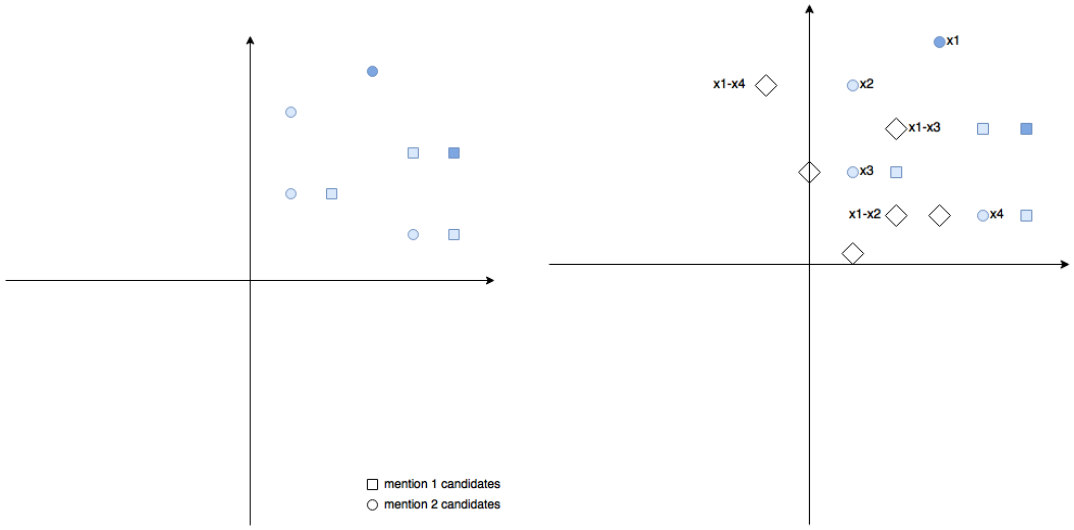
Training

The model is trained as follows. Let m be a named entity mention that refers to the named entity e^m and let E_m be the set of named entity candidates generated for m. We assume $e^m \in E_m$. Then for each $e_i \in E_m$ we create a feature vector $x_i = \langle e_i, m \rangle$ generated from the candidate mention pair. The rank r_i of the vector x_i is equal to 1 if $e_i = e^m$, otherwise $r_i = 0$. In Figure 3.4a we can see an example of two groups of candidate vectors represented by circles and squares. The true entity for each group is colored with darker blue.

In the next step we apply the so-called pairwise transformation. For each (x_i, x_j) pair of the same group (generated from the same mention) and having a different rank score ($r_i \neq r_j$), we compute their difference vector $d_{i,j} = x_i - x_j$, where $i < j$. Note that we discard the inverse of a difference vector, meaning that if we computed $d_{1,3}$ we do not need to consider $d_{3,1}$ as it provides the same information ($d_{1,3} = -d_{3,1}$). For each computed difference vector $d_{i,j}$ we define its label $y_{i,j}$ as positive ($+1$) if $r_i > r_j$ and negative otherwise (-1). In Figure 3.4b we can see the created difference vectors of each group represented as white diamonds.

Note that this may lead to very unbalanced data, as we have only one positive vector. As shown in Figure f2, since x_1 is the feature vector for the true candidate mention pair we obtain only positive difference vectors. To balance the data we randomly inverse them by doing $d_{ij} = -d_{i,j}$ and $y_{i,j} = -y_{i,j}$. In Figure 3.5a we can see the result of this balance operation.

Once we have produced a set of balanced binary difference vectors we just need to separate them by using the SVM algorithm. The computed optimal hyperplane is



(A) Two groups of candidate entity vectors (B) Difference vectors for each group

FIGURE 3.4

defined by the equation $x^T \cdot w = 0$ where the vector w is its normal vector. We can see these two elements in Figure 3.5b. As described in [13], if the data is not linearly separable, then kernel functions can be applied.

Ranking function

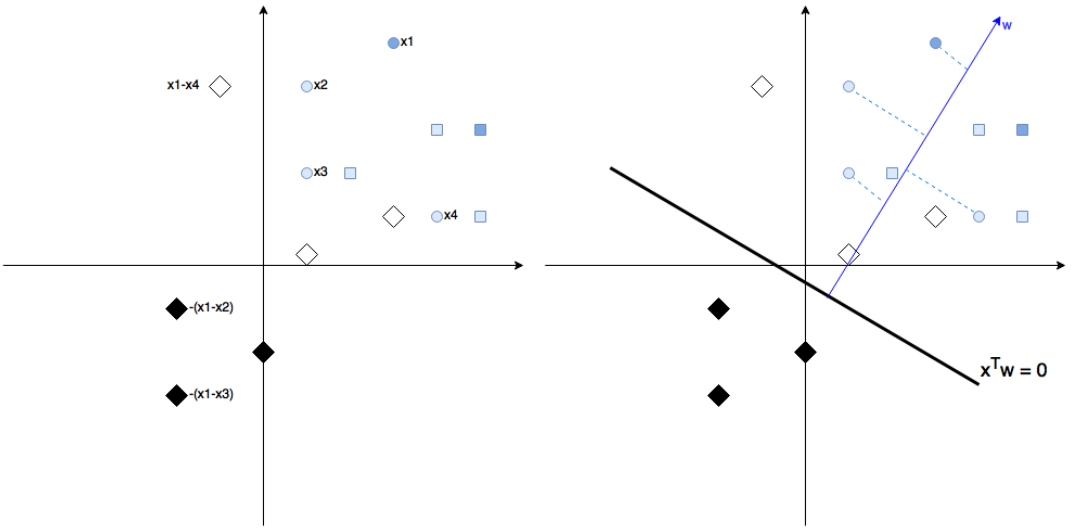
Given the optimal hyperplane $x^T \cdot w = 0$, our ranking function is defined as:

$$f_r(\phi) = \phi^T \cdot w$$

In fact the projection of a vector on the normal vector w of the hyperplane gives us its rank score. The higher the score, the higher the rank. Then, when we want to disambiguate a mention m we do the following. As in the training phase, for each $e_i \in E_m$ we generate the feature vector $x_i = \langle e_i, m \rangle$. Then we simply project them on w by applying the ranking function to each feature vector. The vector x_i that obtains the higher score will be the winner, implying that the candidate ei will be linked to the mention m. The ranking phase is shown in Figures 3.6a and 3.6b.

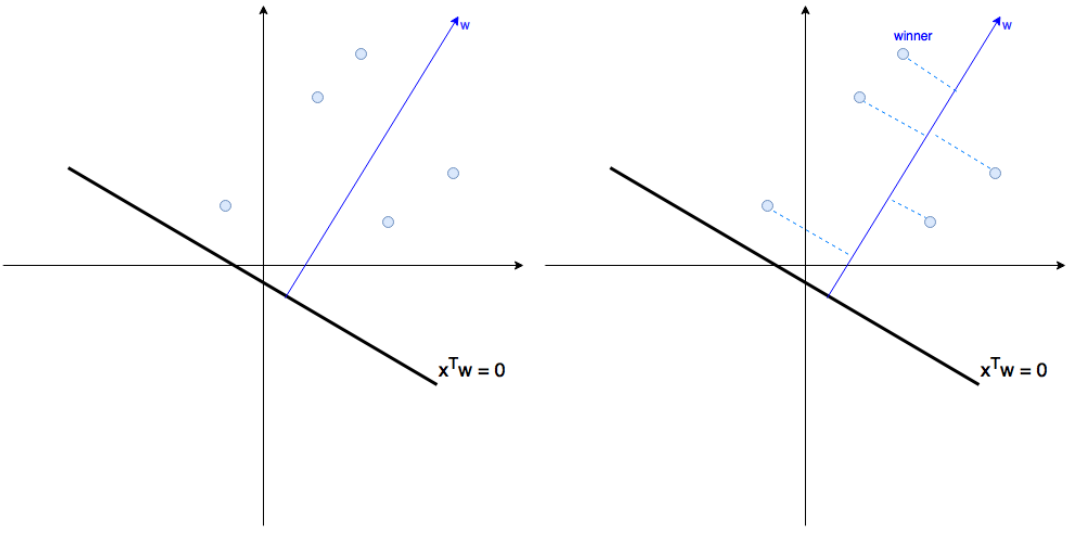
Interpretability

An advantage of the Ranking SVM algorithm is that, once we have computed the optimal vector w composing the ranking function, we can interpret the weights of w. Intuitively, if a weight has a large value then it means that the feature associated to that weight played an important role in establishing the rank difference between two candidates. In other words, the higher the weight the higher the discriminative power of the feature associated to that weight. Similarly if a weight has a large negative value it means that the feature will rank lower an entity that has that feature, that feature will have an high level of inversed discriminative power. If instead the weight is close to zero, it means that the feature is little relevant to the ranking score.



(A) Balanced difference vectors (B) The hyperplane's normal vector, created from the separation of the data

FIGURE 3.5



(A) Candidate vectors for a new mentions (B) Ranking of the candidates by projecting them on w

FIGURE 3.6

One inconvenient with this algorithm, however, is that the ranking score itself is difficult to interpret. Since the determinant features can be either positive or negative, the vector ranked first could win against the others by having an high score, a low one or even negative. Thus, it is not easy to define a level of confidence of the result as it is possible to do with binary classifiers like SVM and to use it, for example, in active learning techniques.

3.5 Features

In this section we describe the features that are used to train the learning to rank algorithm presented in the previous section. The features fall into three categories: string similarity features, contextual features and probability features.

3.5.1 String Similarity Features

The goal of this group of features is to capture the similarity between the mention surface form and the names of a candidate entity, which can be either an ancient author or an ancient work depending on the mention type.

Exact Matching

This kind of features determine if there are exact matches between single words or between groups of words. For example "pliny elder" and "pliny elder" is an exact match but also "pliny elder" and "elder pliny" is considered as such.

In total 3 Exact Matching based features are used (boolean only).

Fuzzy Matching

This kind of features capture such cases where words to be compared are not exactly the same but they are similar enough to be related. They are used between single words or groups of words and thy can be either boolean (if the score is above a certain threshold) or continuous (by returning the score directly). Fuzzy Matching features are based on some metrics defined below.

Normalized Levenshtein Distance (NLD):

The Levenshtein Distance (LD) is, informally, defined as, given two strings s_1 and s_2, the minimum number of insertions, deletions or substitutions of characters that are necessary to transform one string into the other [15]. There is more than one technique to normalize this metric [29], we chose as a normalization factor the sum of the length of the two strings. Finally we inversed the score to have a distance of 1.0 between two equal strings and to be consistent with other metrics used in the features. Our NLD is given by:

$$NLD(s_1, s_2) = 1 - \frac{LD(s_1, s_2)}{|s_1| + |s_2|}$$

Common Initial Letters Ratio (CILR):

This metric is used to capture the amount of initial characters that two strings have in common. Let $CIL(s_1, s_2)$ be the number of initial characters that two strings s_1 and s_2 have in common, then the CILR is defined as:

$$CILR(s_1, s_2) = \frac{CIL(s_1, s_2)}{\min(|s_1|, |s_2|)}$$

Phonetic Similarity (PHS):

This metric is used to capture the phonetic similarity between two strings s_1 and s_2. The *jellyfish*[4] Python library includes three algorithms to convert a string to a normalized phonetic encoding (the literal representation of its pronunciation): American Soundex, Metaphone, the New York State Identification and Intelligence System Phonetic Code (NYSIIS) and Match Rating Approach (MRA). We finally chose NYSIIS as phonetic code, as it improves the Soundex algorithm. Finally we needed a way to compare two phonetic encodings. To do so we decided to apply the Normalized Levenshtein Distance to the encodings to measure their similarity. Thus, let $PHC(s)$ be the phonetic code computed by the NYSIIS algorithm of a string s, then the PHS is defined as:

$$PHS(s_1, s_2) = NLD(PHC(s_1), PHC(s_2))$$

We also used the Match Rating Approach comparison, available in jellyfish as well, to compare two strings encoded with the MRA (codex) algorithm.

In total we defined 11 Fuzzy Match based features (3 continuous and 8 boolean).

Abbreviations and Acronyms

This kind of features deal with various forms of abbreviations and acronyms. Abbreviations can be made by one or multiple words and can be constructed by the first characters of a word or by sparse sequences of characters contained in the word. Acronyms are in general composed by the first character of each word contained in a name or title.

These features are necessary because of the incompleteness of the abbreviation lists in the KB. For Example

In total we have 3 boolean features for abbreviations and 1 boolean feature for acronyms.

3.5.2 Context Features

String similarity features do not provide, in some cases, enough information to disambiguate a named entity mention. This is due to the high level of ambiguity of some mentions and also to the incomplete information of the knowledge base. Therefore it was necessary to identify a context to capture the semantic information

[4] http://jellyfish.readthedocs.io/en/latest/index.html

about a mention that could help its disambiguation. Unfortunately we cannot exploit the usual context features as it is done in state-of-the-art NED systems because of two main reasons. The first reason is that our knowledge base does not contain semantic information (text, link structure, categories tag, ...) about a named entity. The second reason is that the documents in which the mentions appear are often very short texts and thus they provide little contextual information.

Neighboring mentions similarities

A way to improve the disambiguation of an ambiguous mention is to check whether the document contains other mentions that are somehow related to a given candidate entity. For example, if we are trying to disambiguate an ancient work with an ambiguous name, we could take from the KB the entity author of the current entity work candidate and search for a match against author mentions of the same document. In addition we also could check if other mentions match the names of the candidate work as it could be referenced in a less ambiguous form elsewhere in the document. Similarly we can use the same techniques when the candidate entity is an ancient author.

In Figure 3.7 we can see an example of how this information can be used to improve the disambiguation of a work. Surrounded by a square we have a *REFAUWORK* mention we want to disambiguate. In Table 3.5 we saw that in the KB there are 75 works that can be abbreviated "Carm." The blue frame contains the information of the candidate being the true entity. We can see from the dashed blue arrow that there is another mention in the document that matches the candidate, however in this case it is not useful as it does not provide additional information. If we consider the author of the candidate work, represented by the red frame, we can see from the dashed red arrows that it matches two other mentions in the document, providing additional information about the likelihood of the candidate being the correct entity.

The matching of the candidate names and abbreviations with the other mentions contained in the text is based on the string similarity features previously described.

In-title mentions similarities

Another context element we decided to use is the titles of the APh articles where the abstracts were taken from. Similarly to the previous techniques we can check if the work/author candidate names or the names of its author/works are present in the title. One limitation is that not all the documents were provided with a title.

TF-IDF cosine similarity

Despite the fact that our knowledge base does not provide contextual information, there is still a certain number of entities that do have a Wikipedia page, which could be used as a source of contextual information. We decided to use this information because of two reasons. The first was to analyze the impact of such information against the other features. The second is that we believe this kind of information should be used when available and thus we provide an example of how this can be

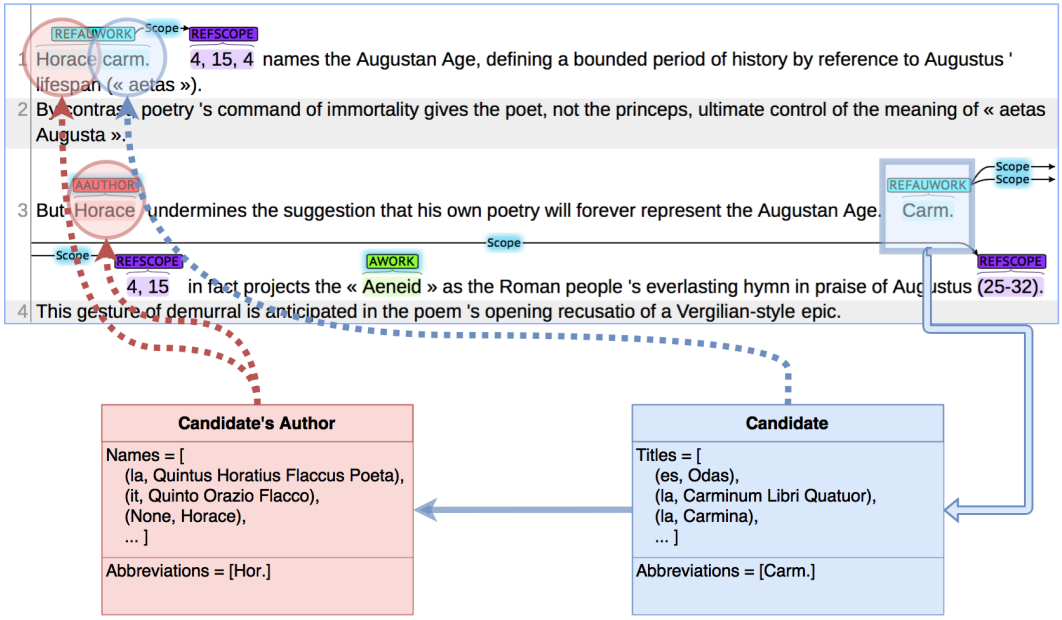


FIGURE 3.7: Example of contextual information used in Context Features

done. To this end, we decided to fetch the text of Wikipedia articles about the ancient authors in our knowledge base, written in various languages (English, Spanish, German, French, Italian).

The problem was, given an entity, how to automatically get its Wikipedia entry. Querying the Wikipedia search engine with the names of the authors is impractical because the names are often incomplete and also it is necessary to deal with Wikipedia ambiguous pages. A better way was to use the CTS URN identifiers of the KB. By using the Perseus Catalog of Greek and Latin literature[5] we were able to map the CTS URN identifiers of some authors to a VIAF[6] (Virtual International Authority File) identifier. Then by querying the Wikidata SPARQL (SPARQL Protocol and RDF Query Language) endpoint we mapped the VIAF identifiers to the Wikidata identifiers. By using the Wikidata HTTP API we mapped the Wikidata identifiers to their Wikipedia page title for each language if available. Once we have the page titles it is easy to get the text from Wikipedia. Finally we obtained the Wikipedia pages for 88 authors, with 3 of them not having a version for each language. The main bottleneck was the the Perseus catalog, which for now includes only a small number of CTS URN and VIAF identifiers mappings.

There are different techniques to compare textual context. As we are dealing with short text documents we decided to use the Term Frequency-Inverse Document Frequency (TF-IDF) as weighting-scheme for words. The weights are computed on the vocabulary extracted from the pre-processed Wikipedia articles mentioned above. For each language we constructed a words-documents matrix containing the TF-IDF weights.

Once the matrices were constructed, we used the cosine similarity to measure the relatedness between a mention's document and the Wikipedia article text, when available, of a candidate author or candidate work's author.

[5]http://catalog.perseus.org
[6]https://viaf.org

3.5.3 Probability Features

Computing and using statistics about the distribution of mentions and candidates is often used in the literature. We identified three different probability measures that capture different aspects of the data, which we explain in the following sections. In general, the effectiveness of these kind of features mainly depends on how much representative is the data from where the distribution is computed. In fact the more entities are represented in the training set, the better these features will generalize. Even if we are conscious of the small size of our data sets, we still decided to test these features. Their impact is discussed in Chapter 4.

Entity prior probability

This probability distribution captures how often an entity appears in the corpus, independently from the mentions that refer to it. The role of this feature is to give more preference to more frequent entities but a the risk of introducing a bias against rarer ones.

To compute it we used the formula described in [10] and it is defined by:

$$p(e) = \frac{count(e) + 1}{M + N}$$

where $count(e)$ is the number of times the entity e is mentioned in the corpus, M is the total number of mentions in the corpus and N is the total number of entities in the KB.

Mention ambiguity

This distribution captures the probability that an entity is referred to by a given mention. Intuitively, the more entities are referenced by a unique surface form, the less the probability that a mention with that form points to one of those entities. We used the definition of [25] given by:

$$p(e|m) = \frac{count(m \to e)}{\sum_{e' \in E_C} count(m \to e')}$$

where $count(m \to e)$ is the number of time that a mention m refers to an entity e and E_C is the set of all entities contained in the corpus.

3.5.4 NIL Features

As already mentioned, we decided to include the NIL entities in the learning process of the Ranking SVM algorithm. To achieve this we need to artificially add the NIL identifier in the candidate set of each mention. The problem is to define how to construct the feature vector given a mention and a NIL entity. Since the NIL entity is not, by definition, in the KB it does not have names or abbreviations that can be used to compute the similarity with the mention. What we need is a way to determine whether a mention is likely to reference an entity which is not in the KB.

As described in the previous chapter, some state-of-art NED systems [26] applied a threshold on the score of the winner entity. If the score was below a certain value the mention was labeled as NIL. Since we want to include the NIL detection in the learning to rank framework we decided to create the features of the NIL candidate vector as follows.

For each mention, once we have created all the feature vectors for all its candidate entities, we collect some statistics from the vectors. As we will explain in the next Chapter, only String Similarity features are used to collect the statistics. For each continuous feature of all the candidates, we extract the maximum value, the average, and we compute the difference between them so as to form three new continuous NIL features. Moreover, for each boolean feature we create a new boolean NIL feature that is set to 1 if no candidate has matched this feature, and to 0 otherwise. As stated in [7] this is equivalent to learning different thresholds to determine whether a given mention sh to a NIL entity.

4 Experiments and Evaluation

In this chapter we describe the main experiments we devised as well as their results. The we evaluate the final system against the baseline. To understand the metrics used in the experiments and in the evaluation we describe them in the first section.

4.1 Evaluation measures

The evaluation of NED systems is usually performed by using 4 metrics: precision, recall, F_1 score and accuracy. To better explain how they are computed we give the definitions of the possible outcomes of the disambiguation of a mention:

- True Positive (TP): In this case the mention is correctly disambiguated and the true entity was present in the KB (non-NIL entity).
- True Negative (TN): In this case the mention is correctly disambiguated and the true entity was not present in the KB (NIL entity).
- False Positive (FP): In this case the mention was disambiguated as a non-NIL entity but the true entity was either a NIL or a non-NIL entity.
- False Negative (FN): In this case the mention was disambiguated as a NIL entity but the true entity was a non-NIL entity.

Then the evaluation measures mentioned above are defined as follows:

$$
\begin{aligned}
Precision &= \frac{TP}{TP + FP} \\
Recall &= \frac{TP}{TP + FN} \\
F_1 &= \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \\
Accuracy &= \frac{TP + TN}{TP + TN + FP + FN}
\end{aligned}
$$

In the case we are not dealing with NIL entities, it can be shown that $Accuracy = Precision = Recall = F_1$. The evaluation measure can be simply seen as the number of mentions correctly disambiguated over the total number of mentions. However if we take into account NIL entities, the $Accuracy$ could be a misleading accuracy measure. As an example consider a data set of 100 mentions where 50 of them are NIL and the other 50 are non-NIL. Assume a NED system disambiguate correctly one non-NIL mention and all the others are disambiguated as NIL. Therefore we have $TP = 1, FP = 0, TN = 50, FN = 49$ leading to $Precision = 100\%, Recall =$

$2\%, F_1 = 39.2\%, Accuracy = 51\%$. We can intuitively understand that these measures capture different aspect of the system performance. $Precision$ and $Recall$ do not take into account how good the system is to detect NIL entities, it is more about non-NIL. $Accuracy$ is more general and includes the capability of linking NIL mentions to NIL entities. Since the main goal here is the extraction of citations, we want to consider $F_1 score$ as it would give us a measure about how well non-NIL entities are detected, which is difficult to understand with $Accuracy$ only.

4.2 Candidate Set Generation

As mentioned in Chapter 3 we describe in this section what represent a match between a mention surface form and a name/title or abbreviation of an entity.

As mentioned in Chapter 3 we describe in this section the functions that define a match between a mention surface and a name/title or abbreviation of an entity. A matched entity will be then added to the candidate set of that mention.

Initially we simply used exact matching. This approach led to poor results in term of recall because of the following reasons: not all names/titles and abbreviations used in the corpus were present in the KB as a possible name of an entity. Furthermore there were cases where the mention exactly match only a part of the name, for example the mention "pliny" does not exactly match the name "pliny elder", which is the normalized form of the name "Pliny the Elder". In addition there were some cases in which the mention was written in a language that was not present in the KB.

To cover the previously described cases we added the following matching functions between mentions and names/titles. To deal with different language versions of a name we used the Normalized Levenshtein Distance (NLD) described in the previous chapter and defined as a match whether the resulted score was above 0.7. Furthermore we added as a match whether there is at least one word of the mention surface form that exactly matches at least one word of the target name. These additions improved the recall for *AAUTHOR* and *AWORK* mentions but for the *REFAUWORK* the results were still poor.

After analyzing the forms of the missed *REFAUWORK* mentions we saw that there were a significant amount of cases in where the mentions were a combination of an author abbreviation and a work abbreviation. To cover this case we added the following match functions. We consider a match if there is at least one word of a name/title that starts with one word of the mention. In addition, we consider a match if there is at least one word of the mention that exactly matches an abbreviation. Furthermore we noticed that some mentions was acronyms of a names/titles and therefore we added as a match whether the mention is an exact acronym of the target name/title.

Finally, with the previously described matching functions, we achieved a reasonable percentage of recall that is shown in Table 4.1. In the table are included the statistics about candidates set sizes of NIL mentions, which do not have a recall score because the correct entity will never be present in the candidates set as it does not exist in the KB.

By inspecting the non-NIL mentions that do not have the correct entity in their set of candidates we discovered some errors that were present in the KB. Such particular

TABLE 4.1: Statistics about the generated candidate sets

	Recall	Min size	Max size	Avg. size
AAUTHOR	441/449 (98.2%)	1	700	71.2
AWORK	221/231 (95.7%)	1	748	98.0
REFAUWORK	413/429 (96.3%)	2	1699	186.2
Total	1075/1109 (96.9%)	1	1699	121.3
NIL	-	0	2800	213.7

cases suggest that the KB need to be improved by adding these very rare forms to the names/titles or abbreviations of the concerned entities. This improvement has, in fact, already started at the time we are writing this report.

In Figure 4.1 we show the impact of the matching functions on the percentage recall and on the average size of the candidates sets.

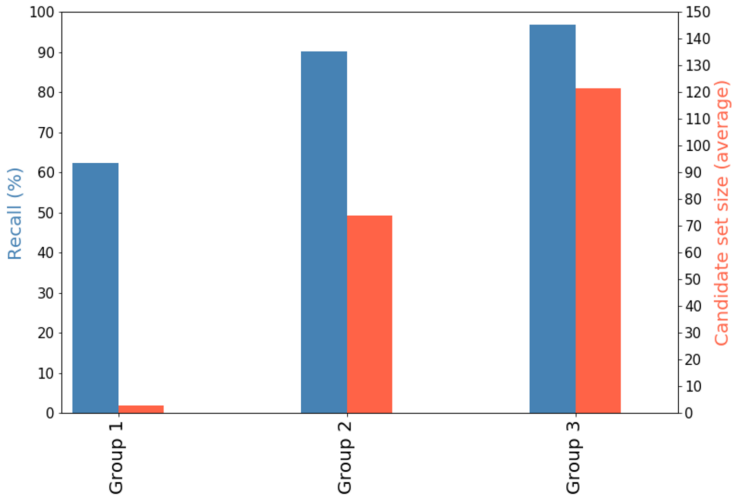


FIGURE 4.1: Recall and average size of the candidates set for each group of similarity functions

4.3 Features Engineering

In this section we describe the main experiments we performed and the modifications we adopted about the features used to train the Ranking SVM algorithm described in the previous chapter. These experiments involving the features initially do not include NIL features. The reason is that NIL features are constructed from the already defined features of the candidates, as described in the previous section. As a consequence we only introduced NIL features when we reached a sufficiently quality of the main features.

4.3.1 Per-type Ranking Function

When analyzing the errors while using the previously described features we noticed that some features that should have been relevant to determine the ranking had

a very low impact. We realized that this was probably due to the fact that these features were good only for a particular type of mentions. For example consider the feature that measure whether the mention surface form is a combination of the candidate work entity title and the name of its author. This feature is never true for mentions of type *AAUTHOR* and *AWORK* because such case appears only in *REFAUWORK* mentions. As a consequence the ranking algorithm learns that this feature, in the majority of the cases, is not relevant to determine the winner candidate and therefore its weight of the ranking vector w will end up to have a small value.

To overcome this fact we decided to separate the features per mention type. This means that, given a feature f, we define the same feature for each type leading to have the features $f_{AAUTHOR\_NO\_SCOPE}$, $f_{AAUTHOR\_SCOPE}$, f_{AWORK} and $f_{REFAUWORK}$. For example, given a mention of type *AWORK* and a candidate entity, only the feature f_{AWORK} is computed, the others are set to zero.

This hypothesis was also confirmed by [7], where the same technique was used. As they claimed, separating the features per mention type is equivalent to define a different ranking function for each type. An advantage of this approach is that it is possible to specify features that are specific for a particular mention type.

Since the surface forms of our mentions have very different structures we decided to experiment whether it would be beneficial to separate the features also by the number of words present in the mention. This is motivated by the fact that the number of words is sometimes an indicator of the nature of a mention. For example the mentions that are composed by more than two or three words will likely not be an abbreviation.

With the introduction of these two degree of separation we had an increase of 8.2% in terms of Accuracy.

4.3.2 Feature Combinations

Inspired by the work of [7] we decided to experiment with feature combinations. The goal of this technique is to capture eventual correlations that could exist between certain features. This is done by computing the logical conjunction of a particular sub-set of features. For example, if we are combining the features as group of two, given the features f_1, f_2, f_3 we can compute the combined features $f_{1,2} = f_1 \wedge f_2$, $f_{1,3} = f_1 \wedge f_3$, $f_{2,3} = f_2 \wedge f_3$. The problem with this approach is that the number of features could easily explode and therefore the size of the combined groups should be limited to 2 or 3 features.

We combined the features in groups of 2. Only the features that come from the same mention type are combined together. Non-boolean features were discretized using two thresholds to quantize the continuous value into three categories: high, medium, low. For example given a feature $f_1 \in [0, 1]$ we introduced three boolean features as follows:

$$f_{1\_high} = \begin{cases} 1 & \text{if } f_1 > 0.7 \\ 0 & \text{otherwise} \end{cases}$$

$$f_{1\_medium} = \begin{cases} 1 & \text{if } 0.3 \leq f_1 \leq 0.7 \\ 0 & \text{otherwise} \end{cases}$$

$$f_{1\_low} = \begin{cases} 1 & \text{if } f_1 < 0.3 \\ 0 & \text{otherwise} \end{cases}$$

We did this experiment only one time as it took more than 10 hours to complete. As it did not show improvements we decided not to use feature combinations.

4.3.3 NIL Features

When we included NIL, the overall performance decreased. Initially we combined all the features of a given set, including Context features and Probability features. By doing this we observed a strong over-fitting in favor of NIL entities. Our hypothesis was that the combined features was too noisy as they included too much information. Finally we decided to combine only String Similarity features to produce the NIL features. With this approach we regained a certain balance between NIL and non-NIL.

4.4 Disambiguation Refinement through Co-occurrence Probability

In general, in our experiments we noted that the true candidate was, if not ranked first, almost always in the first positions of the rankings (90.1% accuracy for the top-10 candidates). This tells us that is difficult for the system to always rank first the correct entity while it is easier to have it in the first top-10 results on an average candidates set size of 121.3.

The above fact motivated us to investigate a method to take advantage of that result. Inspired by the collective ranking techniques described in Chapter 2 we formulated the following assumption. If there are entities that are more related than others, then given a limited number of small sets of candidates we could infer whether there is a configuration of candidates that is more probable to occur. For example, if we have two mentions m_1 and m_2 with respective ordered candidate sets $E_{m_1} = \{e_1\}$ and $E_{m_2} = \{e_2, e_3\}$ and if entities e_1 and e_3 are strongly related while e_1 and e_2 are not, then we would like to choose the configuration $\{m_1 = e_1, m_2 = e_3\}$ instead of $\{m_1 = e_1, m_2 = e_2\}$. In such case we can see that even if e_3 was not the first choice for m_2, by using a collective disambiguation approach we could be able to improve the performance of the system. Another example of a possible configuration of candidates given the top-5 candidates of three mentions is shown in 4.2.

However collective ranking based NED systems relies on the information provided by their KBs to compute the relatedness between two entities but in our case it is, in general, not possible. Therefore we needed a method to compute such similarity. To

define that method we used the co-occurrence probability of the entities extracted from the corpus. Our hypothesis was that, if the entities co-occurrence distribution of the train set is enough representative of the real distribution, then we could exploit this information to perform the collective inference of the most probable configuration of candidates.

Due to its similarity to our approach, we decided to base our model on [14]. Given a possible configuration of candidates $C = \{m_1 = e_1, m_2 = e_2, ..., m_n = e_n\}$ where $n = |M_i|$ is the size of the set of mentions M_i appearing in a document D_i, they compute its score as:

$$score(C) = \frac{1}{\binom{|C|}{2}} \sum_{e_i \neq e_j \in C} r(e_i, e_j) + \frac{1}{|C|} \sum_{e_i \in C} f_r(e_i) \tag{4.1}$$

where $r(e_i, e_j)$ is the a function that measure the relatedness between two entities and $f_r(e_i)$ is a ranking function. As mentioned before we used co-occurrence to model relatedness. We defined relatedness, similarly as done in [1], as $r(e_i, e_j) = p(e_i, e_j) = p(e_j|e_i) \cdot p(e_i)$ where $p(e_j|e_i)$ represents the probability of e_j occurring with e_i and $p(e_i)$ is the entity probability described in the previous chapter. We used the score of our Ranking SVM function for $f_r(e_i)$. The co-occurrence matrix is built by counting the number of documents in which two entities appear together. The matrix is then normalized with respect to the number of their documents.

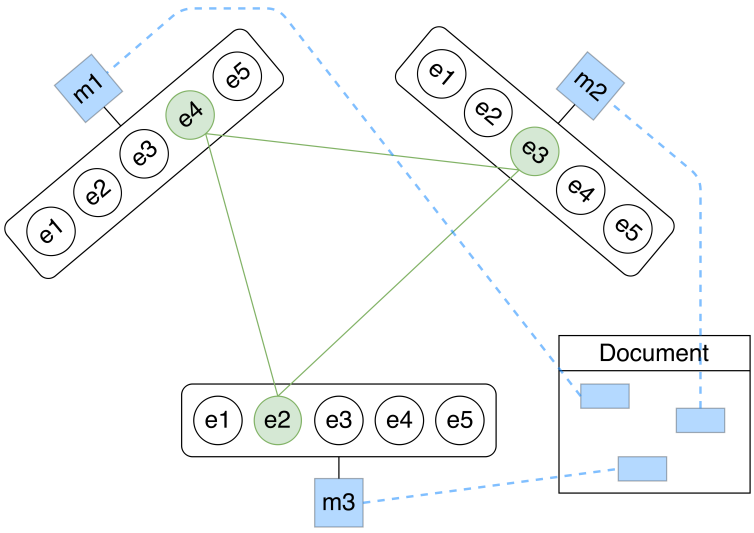


FIGURE 4.2: A possible configuration of candidates (green) given the top-5 candidates of three mentions

As described in Chapter 2 the direct optimization of this problem is NP-hard. In fact, as the number of mentions grows in a document, it takes exponential time to compute the probability of each possible configuration. Despite the intractability of such problem there exist approximate algorithms that allow to compute an optimal solution more efficiently like hill climbing or linear programming techniques [14]. However due to scheduling constraints we did not have sufficient time to adequately explore this direction of research. As a consequence, to test the validity of the solution we decided to apply the collective co-occurrence based disambiguation on documents containing a relatively small number of mentions (less than 6), leading to a sub-set of 59 documents (over a total of 82), and by using the top-5 ranked entity

candidates for each mention. In addition, we inspected the ranking scores to detect the presence of some confidence thresholds above the ones the entity is always true but the results were very noisy, suggesting no information could be used to identify disambiguation candidates that are very likely to constitute the true match.

The first experiments with (4.1) showed poor results. This was mainly due the score contribution in the formula, which tends to give more importance to the winner candidates that, in most cases, we are trying to correct if they are wrong. We tried then to remove the scores from the the objective function. As a consequence, the normalization factor was no more needed leading to the following function:

$$score(C) = \sum_{e_i \neq e_j \in C} r(e_i, e_j)$$

Experiments with (4.4) showed a little improvement. We realized that, too often, configurations containing NIL entities received too high scores. This is mainly due to the fact that NIL entities were included in the co-occurrence matrix and the co-occurrence probability of two NIL entities was relatively high. As a consequence, we decided to define $r(e_{\text{NIL}}, e_i) = r(e_i, e_{\text{NIL}}) = r(e_{\text{NIL}}, e_{\text{NIL}}) = 0$ as in [14]. Finally we obtained an improvement of the 6.8% of accuracy with respect to the score obtained with our best set of features on the sub-set defined above. In conclusion, this method seems to be very promising and it certainly deserves further exploration.

4.5 Evaluation

As mentioned in Chapter 3 the disambiguation module described in [24] is not based on supervised machine learning technique, the whole manually annotated corpus was used to evaluate it as it didn't need a training set. However one the goals of this project was to exploit the annotated data to devise a supervised approach. As a consequence the data needed to be split in a train and test sets. A first split was made to keep a balance between languages as the data was used also to train the Extractor of Citation Components module used in [24]. Then, after an analysis of the distribution of mention types (including NIL entities) we noticed that some of them, in particular NIL and *REFAUWORK* mentions were quite unbalanced. Since the size of our data sets is quite small, we decided to generate another train/test split to improve the balance between types. The new distribution is shown in Table 4.2.

TABLE 4.2: Mention types distributions over train and test set

	Train set	Test set
AAUTHOR	329 (33.5%)	119 (38.5%)
AWORK	157 (16.0%)	73 (23.6%)
REFAUWORK	348 (35.5%)	81 (26.2%)
NIL	147 (15.0%)	36 (11.7%)
Total	981	309

As explained in 3.1.2, when disambiguating canonical references we ignore the passage scope of the citation because parsing the sequence of numbers representing the

scope and attaching it to the identifier of the disambiguated entity is a step that is independent from the disambiguation task. Therefore in the evaluation phase, when an entity mention of type *AAUTHOR*, *AWORK* or *REFAUWORK* is linked to multiple *REFSCOPE* mentions representing more than one canonical citation (e.g. « *Prometeo encadenado » de Esquilo, especialmente en los v. 399-402 = 408-411 y 425-435.*), we count this mention only once instead of for each different scope. This was mainly due to avoid bias represented by mentions linked to many passages. For example, if a mention is linked to ten passages, then linking that mention to an incorrect entity should count as one error and not as ten. In Table 4.3 we can see the distribution of the test set according to the above definition. We include as a separated type the authors and works involved in Scope relations.

TABLE 4.3: Mention types distributions over train and test set without counting multiple Scope relations

	Train set	Test set
AAUTHOR	277 (34.2%)	109 (42.6%)
AWORK	111 (13.7%)	53 (20.7%)
AAUTHOR-scope	27 (3.3%)	6 (2.3%)
AWORK-scope	30 (3.7%)	9 (3.5%)
REFAUWORK-scope	232 (28.6%)	49 (19.1%)
NIL	133 (16.4%)	30 (11.7%)
Total	810	256

4.5.1 Results

In Table 4.4 we show the results of the baseline, described in 3.1.3, and our chosen model. Even if the margin is little, we can see that our model performs better than the baseline.

TABLE 4.4: Disambiguation scores for Group 3 and the Baseline

	Precision	Recall	F_1 score	Accuracy
Baseline	68.07%	77.49%	72.47%	65.23%
Model	**73.89%**	**81.75%**	**77.62%**	**71.88%**

If we look at Table 4.5 we can see the Baseline is better in detecting NIL entities. The main problem remains, for both models, the disambiguation of *REFAUWORK* mentions. Even if our model performs slightly better, this kind of mentions are still very difficult to label and they have an important impact on the overall performance of the system.

We analyze here the impact of the three main groups of features described in Chapter 3. Group 1 represents the String Similarity features described in 3.5.1, group 2 represents group 1 features plus the Context features described in 3.5.2 and group 3 represents group 2 features plus the Probability features described in 3.5.3. All the groups use the per-type division explained in 4.3.1 and the NIL features (3.5.4) are included as well.

In Table 4.6 we can see that in general, even if the results are quite similar between the groups, the group 3 containing all the categories of features performs slightly

TABLE 4.5: Per-type Accuracy for Group 3 and the Baseline

	Baseline	Model
AAUTHOR	84.40%	91.74%
AWORK	71.70%	81.13%
AAUTHOR-scope	16.67%	50.00%
AWORK-scope	66.67%	77.78%
REFAUWORK-scope	24.49%	32.65%
NIL	60.00%	50.00%

better than the others. Surprisingly the introduction of Context features to the String Similarities features had a negative impact on Precision causing a lower F-score. This can be caused by the fact that Context features involving other mentions in the document suffer from the same high ambiguity problem that affects String similarity features. Another possible reason is the small percentage of authors having an associated Wikipedia document, used to compute textual similarity with mention documents, causing a bias against entities not having such extra information.

TABLE 4.6: Disambiguation scores for each group of features

	Precision	Recall	F_1 score	Accuracy
Group 1	**75.02%**	78.80%	76.86%	68.36%
Group 2	72.59%	80.45%	76.32%	71.48%
Group 3	73.89%	**81.75%**	**77.62%**	**71.88%**

Table 4.7 contains, for each group, the Accuracy score breakdown for each type of mention. NIL entities are included as a separate type. In this table we can see that group 1 does not perform well with AWORK mention while it is the best in labeling NIL entities. Group 2 is not very good in the disambiguation of scope-AAUTHOR mentions, which in general represent opus maximum kind of mentions. Therefore we decided to choose the Group 3 as the final model, which we compared against the baseline. The total number of features of the final model is 484.

TABLE 4.7: Per-type Accuracy for each features group

	Group 1	Group 2	Group 3
AAUTHOR	90.83%	91.74%	91.74%
AWORK	64.15%	81.13%	81.13%
AAUTHOR-scope	50.00%	33.33%	50.00%
AWORK-scope	77.78%	77.78%	77.78%
REFAUWORK-scope	28.57%	32.65%	32.65%
NIL	60.00%	50.00%	50.00%

4.5.2 Error Analysis

By inspecting the results we identified at least three main sources of error.

The first is that in some cases, the surface form of a mention was not present in the KB attributes of the entity it is referring to. Sometimes it was a special abbreviation

that was not related with any name variation. As a consequence the generation of the candidates did not include the true entity in the set of the mention, leading to the impossibility of disambiguating that mention. It is the case of the mention "Briefen" which references the work "Epistulae" of "C. Sollius Apollinaris Sidonius". In the KB we have only "Epistulae" as work titles and "epist." and "ep." as work abbreviations.

Another problem was caused, in some cases, by the use of Probabilistic features. For example there was a case where the mention was "carm." and the system chose the work "Carmina" of "Sollius Apollinaris Sidonius" instead of the true work "Carmina" of "Paulinus Nolanus". The feature vectors of the two candidates had exactly the same score except for the probability that the mention refers to that entity. The probability that "carm." refers to the true work was zero because it did not appear in the train set, while the other one had 0.571. Since this probability had an high impact in determining the true entity in the learning phase, its weight was quite elevated (10.977) and therefore the winner entity gained by a difference of 6.273 with a total score of -14.509. This fact evidences the importance of having a training set that generalizes well, where as many entities in the KB as possible are represented.

Some errors were caused by the introduction of NIL entities. Too often NIL entities are preferred by the system. We observed that in situations where there was a weak relationship between the mention surface form and the names of the true entity, the true entity did not reach a sufficient score to win against the NIL entity. We call a weak relationship when only few features are activated between the mention and the candidate names/abbreviations. For example the mention "Donato, Vita Verg." has a particular structure that matches only a part of the author names ("Aelius Donatus") and a combined abbreviation of the title ("Vita Vergilii"). Since this form was not present in the training set, the weight of the feature matching this case was relatively small. On the other hand all the candidates did not match with any feature this particular form causing the activation of the NIL features. Therefore the NIL entity reached the highest score despite the match with the true entity.

In Table 4.8 we can see that in the top-10 positive features, 5 are NIL features. The weights are normalized. This means that in the training set they were useful to detect NIL entities. The problem is that these high NIL features, when we are ranking the candidates for a mention, tend to increase the score of the NIL candidate more easily than for the others. It seems thus that the system is over-fitting the NIL entities. Table 4.9 seems to suggest the same phenomena.

The features starting with NIL_MAX, NIL_AVG and NIL_MAX-AVG are respectively the maximum, the average and the difference between the two of the values of all the candidates of the feature following that prefix. Features starting with R, W and ANS are respectively features specific to mention types *REAFUWORK*, *AWORK* and *AAUTHOR* (without scope). The following prefixes ss, cxt or prob refer respectively to String Similarity features, Context features and Probability features.

We interpret here some relevant non-NIL features. We can observe that the two first positive features are about fuzzy matching between work titles containing 3 or more words. We can interpret this as the fact that this form of titles has a low level of ambiguity and that variations of such titles share minor differences that can be captured by fuzzy matching. The feature *R_prob_e_given_m* is the probability mentioned in the above example. The feature *W_cxt_om_work_match_nb* represent how often a work mention matches another work mention in the same document, suggesting

that this kind of mentions are often cited more than once in a document. The feature *ANS_ss_2w_fuz_init_match* tells us that 2-word author mentions not involved in scope relations often share the initial letters of both words with the words of a name of the true entity.

TABLE 4.8: The top-10 most weighted positive features

	Feature name	Weight
1	W_ss_3w_fuz_match_max	0.347
2	W_ss_3+w_fuz_match_max	0.265
3	NIL_MAX_W_ss_1w_fuz_init_match_max	0.233
4	NIL_AVG_W_ss_2w_fuz_match_max	0.224
5	NIL_AVG_W_ss_1w_fuz_init_match_max	0.163
6	R_prob_e_given_m	0.158
7	NIL_NO_R_ss_w_1w_abbr_match_nwords	0.148
8	W_cxt_om_work_match_nb	0.140
9	ANS_ss_2w_fuz_init_match	0.134
10	NIL_NO_W_ss_2w_ex_match_nwords	0.123

TABLE 4.9: The top-10 most weighted negative features

	Feature name	Weight
1	NIL_MAX-AVG_W_ss_2w_fuz_match_max	-0.207
2	NIL_MAX_W_ss_2w_fuz_init_match_max	-0.184
3	NIL_AVG_ANS_ss_1w_fuz_match_max	-0.148
4	NIL_NO_W_ss_2w_fuz_init_match	-0.137
5	NIL_MAX_W_ss_1w_fuz_match_max	-0.128
6	NIL_NO_R_ss_w_1w_abbr_match	-0.123
7	NIL_NO_R_ss_w_1w_abbr_match_nwords_sparse	-0.122
8	NIL_AVG_W_ss_1w_fuz_match_max	-0.118
9	NIL_NO_R_ss_a_1w_abbr_match_nwords_sparse	-0.114
10	W_ss_3+w_fuz_init_match_max	-0.102

5 Conclusion & Further Work

We built a model that disambiguates mentions of ancient authors and works against a specific closed KB. Our model is based on the Learning To Rank framework and in particular on the Ranking SVM algorithm. The model combines different type of features based on string similarities, contextual features and probabilistic features. Our method improves the baseline, which is based only on string matching. We also showed that co-occurrence probability of entities extracted from the annotated data can lead to a substantial further improvement of the performances/accuracy.

As a future work, we consider that augmenting the amount of annotated data should be a priority, especially for applying supervised techniques. Increasing the number of documents and including longer ones will increase the effectiveness of approaches based on probability distributions extracted from the data and on document similarity. Note that the use of longer documents implies a re-definition of the scope of the context. If it is too large it may be more suitable to define a smaller portion of the document as the textual context for a specific mention. In addition, generative models, instead of discriminative, could be investigated to extract entity distributions from the data.

Another improvement that techniques based on document similarity could benefit from, is continuing to improve the KB by increasing the number of authors having semantic information like textual description (e.g. from Wikipedia) or tags and also to extend the covering to works.

Another aspect that could be improved in our model is the detection of NIL entities. Their inclusion in the Learning To Rank framework had a negative impact on the overall performance and it was difficult to control. Therefore it would be suitable to define other methods to deal with this type of entities that can include tuning parameters. For example we could use a separate binary classifier to decide if a mention refers to a NIL entity or not.

A possible future investigation could be researching how to apply the collective disambiguation based on entities co-occurrence probability to documents having an elevated number of mentions and candidates by exploring suitable approximate algorithms.

Another research could investigate more on NED techniques applied to micro-blogs like tweets and therefore to take advantage from collaborative methods. The reason is that this specific task, as it is presented now in this report, shares some similarities with the micro-blog task like, for example, the shortness of the documents and the high level of ambiguity of the mentions. Since NED on micro-blogs is an emerging task, more potentially useful techniques will be available soon in the future.

Bibliography

[1] Marc Bron, Krisztian Balog, and Maarten De Rijke. "Related Entity Finding Based on Co-Occurrence". In: *Text REtrieval Conference* (2010). ISSN: 1048776X.

[2] Razvan Bunescu and Marius Pasca. "Using Encyclopedic Knowledge for Named Entity Disambiguation". In: *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics* April (2006), pp. 3–7.

[3] Zheng Chen and Heng Ji. "Collaborative Ranking: A Case Study on Entity Linking". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing* (2011).

[4] Zheng Chen et al. "CUNY-BLENDER TAC-KBP2010 Entity Linking and Slot Filling System Description". In: *Proc. Text Analysis Conference (TAC 2010)* (2010).

[5] Silviu Cucerzan. "Large-Scale Named Entity Disambiguation Based on Wikipedia Data". In: *EMNLP-CoNLL 2007*. 2007, 708–716. ISBN: 9788890354175. URL: http://www.aclweb.org/anthology/D/D07/D07-1074.

[6] Leon Derczynski et al. "Analysis of named entity recognition and linking for tweets". In: *Information Processing and Management* (2015). ISSN: 03064573. DOI: 10.1016/j.ipm.2014.10.006.

[7] Mark Dredze et al. "Entity Disambiguation for Knowledge Base Population". In: (2010), pp. 277–285.

[8] Y Guo et al. "Microblog entity linking by leveraging extra posts". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. 2013. ISBN: 9781937284978.

[9] Xianpei Han. "Collective Entity Linking in Web Text : A Graph-Based Method". In: *Sigir* (2011). DOI: 10.1145/2009916.2010019.

[10] Xianpei Han and Le Sun. "A Generative Entity-Mention Model for Linking Entities with Knowledge Base". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies* (2011).

[11] Taher H. Haveliwala. "Topic-sensitive PageRank". In: *Proceedings of the eleventh international conference on World Wide Web - WWW '02*. 2002. ISBN: 1581134495. DOI: 10.1145/511446.511513.

[12] Johannes Hoffart et al. "Robust Disambiguation of Named Entities in Text". In: *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*. 2011. ISBN: 978-1-937284-11-4.

[13] Thorsten Joachims. "Optimizing search engines using clickthrough data". In: *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '02*. 2002. ISBN: 158113567X. DOI: 10.1145/775066.775067.

[14] Sayali Kulkarni et al. "Collective annotation of Wikipedia entities in web text". In: *Sigkdd 2009* (2009). DOI: 10.1145/1557019.1557073.

[15] Vladimir I. Levenshtein. "Binary codes capable of correcting deletions, insertions, and reversals". In: *Soviet Physics Doklady* (1966). ISSN: 00385689. DOI: citeulike-article-id:311174.

[16] Xiaohua Liu et al. "Entity Linking for Tweets". In: *Acl '13* (2013).

[17] Xiaohua Liu et al. “Joint Inference of Named Entity Recognition and Normalization for Tweets”. In: *Jeju, Republic of Korea* (2012).

[18] Edgar Meij, Wouter Weerkamp, and Maarten de Rijke. “Adding semantics to microblog posts”. In: *Proceedings of the fifth ACM international conference on Web search and data mining - WSDM '12*. 2012. ISBN: 9781450307475. DOI: 10.1145/2124295.2124364.

[19] Rada Mihalcea and Andras Csomai. “Wikify!: linking documents to encyclopedic knowledge”. In: *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management* (2007). ISSN: 15411672. DOI: 10.1145/1321440.1321475.

[20] David Milne and Ian H Witten. “An Effective , Low-Cost Measure of Semantic Relatedness Obtained from Wikipedia Links”. In: *Artificial Intelligence* (2007). DOI: 10.1.1.165.4954.

[21] David Milne and Ian H. Witten. “Learning to link with wikipedia”. In: *Proceeding of the 17th ACM conference on Information and knowledge mining - CIKM '08*. 2008. ISBN: 9781595939913. DOI: 10.1145/1458082.1458150.

[22] Lawrence Page et al. “The PageRank Citation Ranking: Bringing Order to the Web”. In: *World Wide Web Internet And Web Information Systems* (1998). ISSN: 1752-0509. DOI: 10.1.1.31.1768.

[23] Fabian Pedregosa and G Varoquaux. *Scikit-learn: Machine learning in Python*. 2011. ISBN: 9781783281930. DOI: 10.1007/s13398-014-0173-7.2.

[24] Matteo Romanello. “From Index Locorum to Citation Network: an Approach to the Automatic Extraction of Canonical References and its Applications to the Study of Classical Texts”. PhD thesis. King’s College London, 2015. URL: http://hdl.handle.net/11858/00-1780-0000-002A-4537-A.

[25] Wei Shen, Jianyong Wang, and Jiawei Han. “Entity linking with a knowledge base: Issues, techniques, and solutions”. In: *IEEE Transactions on Knowledge and Data Engineering* 27.2 (Feb. 2015), pp. 443–460.

[26] Wei Shen et al. “Linking named entities in Tweets with knowledge base via user interest modeling”. In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '13* (2013). DOI: 10.1145/2487575.2487686.

[27] M Sozio and a Gionis. “The community-search problem and how to plan a successful cocktail party”. In: *Proceedings Of The Acm Sigkdd International Conference On Knowledge Discovery And Data Mining* (2010). DOI: 10.1145/1835804.1835923.

[28] Partha Pratim Talukdar and Fernando Pereira. “Experiments in Graph-Based Semi-Supervised Learning Methods for Class-Instance Acquisition”. In: *Acl* (2010).

[29] Li Yujian and Liu Bo. “A normalized Levenshtein distance metric”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2007). ISSN: 01628828. DOI: 10.1109/TPAMI.2007.1078.

[30] Wei Zhang et al. “I2R-NUS-MSRA at TAC 2011: Entity Linking”. In: *Proceedings of the 2011 Text Analysis Conference (TAC-2011)* (2011). URL: http://www.nist.gov/tac/publications/2011/participant.papers/NUSchime.proceedings.pdf.

[31] X Zhu and Z Ghahramani. “Learning from labeled and unlabeled data with label propagation”. In: *Neuroscience* (2002). DOI: 10.1109/IJCNN.2002.1007592.